PHP MVC With Database (Very Simple Beginner Example)

code-boxx.com/simple-php-mvc-example

Welcome to a tutorial and example of an MVC application with PHP and MySQL. You have probably heard of MVC everywhere in different flavors – Framework, design pattern, concept, architecture, and much more. So just what is it? What does it do and what does it mean? How do we build a PHP application based on MVC!?

MVC stands for "model, view, controller". When it comes to developing an MVC PHP-MYSQL web application:

- **Model** – Refers to the data structure. In this case, the database.
- **View –** Refers to the user interface. The HTML and CSS.
- **Controller –** The "middleman" doing the processing. Accepts input from the view, and works with the model. Self-explanatory, the PHP scripts and libraries themselves.

Yep, it's that simple actually. But as some tech people love to do, they make simple things look difficult. Let us walk through a simple example in this guide – Read on!

ⓘ I have included a zip file with all the source code at the start of this tutorial, so you don't have to copy-paste everything... Or if you just want to dive straight in.

QUICK SLIDES

TABLE OF CONTENTS

DOWNLOAD & NOTES

Firstly, here is the download link to the example code as promised.

QUICK NOTES

- Create a dummy database and import `0-users.sql`.
- Change the database settings to your own in `1-model.php`.
- Launch `3-view.html` in your browser.

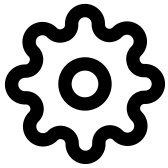
If you spot a bug, feel free to comment below. I try to answer short questions too, but it is one person versus the entire world... If you need answers urgently, please check out my list of websites to get help with programming.

EXAMPLE CODE DOWNLOAD

Click here to download the source code, I have released it under the MIT license, so feel free to build on top of it or use it in your own project.

PHP MVC EXAMPLE WITH MYSQL

All right, let us now get started with the example of a simple MVC application with PHP and MySQL – Doing a search for users.



0) DUMMY TABLE & DATA

0-users.sql

```
CREATE TABLE `users` (
  `id` bigint(20) NOT NULL,
  `name` varchar(255) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

ALTER TABLE `users`
  ADD PRIMARY KEY (`id`),
  ADD KEY `name` (`name`);

INSERT INTO `users` (`id`, `name`) VALUES
  (1, 'John Doe'),
  (2, 'Jane Doe'),
  (3, 'Rusty Terry'),
  (4, 'Peers Sera'),
  (5, 'Jaslyn Keely');
```

First, here is the dummy users table that we will be using. Very straightforward with only the user ID and name.

1) MODEL (DATABASE)

1-model.php

```
<?php
class DB {
  // (A) CONNECT TO DATABASE
  public $error = "";
  private $pdo = null;
  private $stmt = null;
  function __construct () {
    try {
      $this->pdo = new PDO(
        "mysql:host=".DB_HOST.";dbname=".DB_NAME.";charset=".DB_CHARSET,
        DB_USER, DB_PASSWORD, [
          PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
          PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC
        ]
      );
    } catch (Exception $ex) { exit($ex->getMessage()); }
  }

  // (B) CLOSE CONNECTION
  function __destruct () {
    if ($this->stmt!==null) { $this->stmt = null; }
    if ($this->pdo!==null) { $this->pdo = null; }
  }

  // (C) RUN A SELECT QUERY
  function select ($sql, $cond=null) {
    $result = false;
    try {
      $this->stmt = $this->pdo->prepare($sql);
      $this->stmt->execute($cond);
      $result = $this->stmt->fetchAll();
      return $result;
    } catch (Exception $ex) {
      $this->error = $ex->getMessage();
      return false;
    }
  }
}

// (D) DATABASE SETTINGS - CHANGE TO YOUR OWN!
define("DB_HOST", "localhost");
define("DB_NAME", "test");
define("DB_CHARSET", "utf8");
define("DB_USER", "root");
define("DB_PASSWORD", "");
```

Model refers to the data structure. So in the case of a PHP-MYSQL application, our "model" will be a database class that deals with all the "database stuff" – Connecting to the database, managing the connection, and doing SQL queries.

2) CONTROLLER (PROCESS)

2-controller.php

```php
<?php
// (A) DATABASE CONNECTION
require "1-model.php";
$DB = new DB();

// (B) SEARCH FOR USERS
$results = $DB->select(
  "SELECT * FROM `users` WHERE `name` LIKE ?",
  ["%{$_POST["search"]}%"]
);

// (C) OUTPUT RESULTS
echo json_encode(count($results)==0 ? null : $results);
```

The controller is a “middleman” that bridges the user input and model. In this case, it will be an endpoint that accepts a `$_POST["search"]` from the user, uses the model (database) to search, and return the results.

3) VIEW (USER INTERFACE)

3-view.html

```html
<!-- (A) SEARCH JAVASCRIPT -->
<script>
function doSearch () {
  // (A1) GET SEARCH TERM
  var form = document.getElementById("mySearch"),
      data = new FormData(form);

  // (A2) AJAX - USE HTTP:// NOT FILE://
  fetch("2-controller.php", { method:"POST", "body":data })
  .then(res => res.json()).then((res) => {
    let results = document.getElementById("results");
    results.innerHTML = "";
    if (res !== null) { for (let r of res) {
      results.innerHTML += `<div>${r.id} - ${r.name}</div>`;
    }}
  });
  return false;
}
</script>

<!-- (B) SEARCH FORM -->
<form id="mySearch" onsubmit="return doSearch()">
  <input type="text" name="search" required/>
  <input type="submit" value="Search"/>
</form>

<!-- (C) SEARCH RESULTS -->
<div id="results"></div>
```

Finally, the view should be pretty self-explanatory. It is nothing more than HTML, CSS, Javascript to create the user interface.

USEFUL BITS & LINKS

That's it for the example, and here are a couple more bits that may be useful to you.

THE "WRONG" WAY

4-not-good.php

```
<!-- (A) SEARCH FORM -->
<form method="post">
  <input type="text" name="search" required/>
  <input type="submit" value="Search"/>
</form>

<!-- (B) SEARCH + SHOW RESULTS -->
<div id="results"><?php
  if (isset($_POST["search"])) {
    // (B1) DATABASE SETTINGS - CHANGE TO YOUR OWN!
    define("DB_HOST", "localhost");
    define("DB_NAME", "test");
    define("DB_CHARSET", "utf8");
    define("DB_USER", "root");
    define("DB_PASSWORD", "");

    // (B2) CONNECT TO DATABASE
    $pdo = new PDO(
      "mysql:host=".DB_HOST.";dbname=".DB_NAME.";charset=".DB_CHARSET,
      DB_USER, DB_PASSWORD, [
        PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
        PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC
      ]
    );

    // (B3) SEARCH
    $stmt = $pdo->prepare("SELECT * FROM `users` WHERE `name` LIKE ?");
    $stmt->execute(["%{$_POST["search"]}%"]);
    $results = $stmt->fetchAll();

    // (B4) OUTPUT
    if (count($results)>0) { foreach ($results as $r) {
      echo "<div>{$r["id"]} - {$r["name"]}</div>";
    }}
  }
?></div>
```

For the guys who are still confused, maybe a negative example of "not good MVC" will help – Putting all the MVC components into a single script.

THE CAVEAT – GOOD & BAD?

Actually, there is absolutely nothing wrong with putting everything into a single script; It is much more convenient and easier for small projects to do so. But if you think of it in "actual MVC terms", the database (model), the retrieval of data (controller), and display of data (view) are all in one script.

That is not what we want in MVC. The whole idea of MVC is to have a clear-cut separation of concern, have individual scripts that deal with each component. I.E. Scripts that deal with the data model should be in an individual script, controller in another, and the HTML view in yet another.

NOT THE ONLY WAY

I can hear the "expert code troll ninjas" scream – This is not MVC! This is just normal programming! MVC should be in the format of `site.com/MODEL/ AND site.com/VIEW/ AND site.com/CONTROLLER/` . Well, feel free to stick with your "fixed idea of how an MVC framework should be". That is not wrong, but please come up with better arguments.

Smart code ninjas should know better – MVC is an abstract idea, it is a framework. How you interpret and design your system structure around it is entirely up to you. This tutorial is only a simple example of the endless possibilities; Even the "wrong example" above is MVC, it's just that all 3 components co-exist in one single script.

LINKS & REFERENCES

Model–view–controller – Wikipedia

INFOGRAPHIC CHEAT SHEET

PHP MySQL MVC Example (Click To Enlarge)

THE END

Thank you for reading, and we have come to the end of this guide. I hope that it has helped you to better understand what MVC is, and if you have anything to add to this guide, please feel free to comment below. Good luck and happy coding!